# Assignment 3 - Part C

Generated by Doxygen 1.12.0

Chapter 1

# Hierarchical Index

## 1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

# Chapter 2

# Class Index

## 2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

# Chapter 3

# File Index

## 3.1 File List

Here is a list of all documented files with brief descriptions:

# Chapter 4

# Class Documentation

## 4.1 Client Class Reference

Represents a client that connects to a Redis server and sends commands in RESP format (GET, SET, DEL)

#### Public Member Functions

- Client (const std::string &host, const std::string &port)

  *Constructs a new Client object and connects to the server at the specified host and port.*
- void send_command (const std::string &command)

  *Sends a command to the server in RESP format and reads the server's response.*

### 4.1.1 Detailed Description

Represents a client that connects to a Redis server and sends commands in RESP format (GET, SET, DEL)

The client connects to the server and sends commands in RESP format. It serializes the commands into RESP format. The client can send GET, SET, and DEL commands to the server. The client runs an interactive REPL loop where the user can input commands to send to the server. The client reads the server's response and displays it to the user.

### 4.1.2 Constructor & Destructor Documentation

#### 4.1.2.1 Client()

```
Client::Client (
            const std::string & host,
            const std::string & port)  [inline]
```

Constructs a new Client object and connects to the server at the specified host and port.

**Parameters**

| | |
|---|---|
| *host* | The host address of the server |
| *port* | The port number of the server |

### 4.1.3 Member Function Documentation

#### 4.1.3.1 send_command()

```
void Client::send_command (
            const std::string & command)  [inline]
```

Sends a command to the server in RESP format and reads the server's response.

**Parameters**

| | |
|---|---|
| *command* | The command to send to the server |

Serialize the command into RESP format

Send the serialized command to the server

Read the server's response and display it to the user

The documentation for this class was generated from the following file:

- src/client.cpp

## 4.2 Server Class Reference

Represents a server that listens for incoming connections.

**Public Member Functions**

- Server (short port, std::unique_ptr< StorageEngine > storageEngine)

  *Constructs a new Server object with the specified port and storage engine.*
- void run ()

  *Runs the server.*

### 4.2.1 Detailed Description

Represents a server that listens for incoming connections.

The server class listens for incoming connections from clients and creates a new session for each client. The server uses an acceptor to accept connections and an io_context to handle asynchronous operations. The server passes a reference to the storage engine to each session to process commands.

### 4.2.2 Constructor & Destructor Documentation

#### 4.2.2.1 Server()

```
Server::Server (
            short port,
            std::unique_ptr< StorageEngine > storageEngine)  [inline], [explicit]
```

Constructs a new Server object with the specified port and storage engine.

**Parameters**

| | |
|---|---|
| *port* | The port number to listen on |
| *storageEngine* | The storage engine used to process commands |

### 4.2.3 Member Function Documentation

#### 4.2.3.1 run()

```
void Server::run ()  [inline]
```

Runs the server.

This function runs the server by starting the io_context event loop.

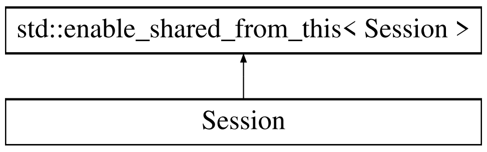
The documentation for this class was generated from the following file:

- src/server.cpp

## 4.3 Session Class Reference

Represents a session with a client.

Inheritance diagram for Session:



**Public Member Functions**

- Session (tcp::socket socket, StorageEngine &storageEngine)

  *Constructs a new Session object with the specified socket and storage engine.*
- void **start** ()

  *Starts the session by reading commands from the client.*

### 4.3.1 Detailed Description

Represents a session with a client.

The session class handles reading commands from the client, parsing the commands in RESP format, processing the commands, and sending the response back to the client. The session class uses a reference to the storage engine to process the commands.

### 4.3.2 Constructor & Destructor Documentation

#### 4.3.2.1 Session()

```
Session::Session (
             tcp::socket socket,
             StorageEngine & storageEngine)  [inline], [explicit]
```

Constructs a new Session object with the specified socket and storage engine.

**Parameters**

| | |
|---|---|
| *socket* | The socket used to communicate with the client |
| *storageEngine* | The storage engine used to process commands |

The documentation for this class was generated from the following file:

- src/server.cpp

## 4.4 StorageEngine Class Reference

A class to represent a disk-persistent key-value store.

```
#include <storage_engine.h>
```

### Public Types

- enum class INITIALIZATION_MODE { CREATE = 0 , OPEN = 1 }

  *An enumeration to specify the initialization mode of the database.*

### Public Member Functions

- StorageEngine (std::string directory, int num_bins, INITIALIZATION_MODE mode=INITIALIZATION_MODE::CREATE)

  *Constructs a new StorageEngine object.*
- ∼**StorageEngine** ()

  *Destroys the StorageEngine object.*
- void insert (std::string key, std::string value)

  *Inserts a key-value pair into the key-value store.*
- bool erase (std::string key)

  *Erases a key-value pair from the key-value store.*
- std::optional< std::string > get (std::string key)

  *Retrieves the value of a key from the key-value store.*

### 4.4.1 Detailed Description

A class to represent a disk-persistent key-value store.

This class provides a class to represent a disk-persistent key-value store. The key-value store stores key-value pairs in a persistent hash map. The class provides an interface to insert, erase, and get key-value pairs from the key-value store.

### 4.4.2 Member Enumeration Documentation

#### 4.4.2.1 INITIALIZATION_MODE

```
enum class StorageEngine::INITIALIZATION_MODE [strong]
```

An enumeration to specify the initialization mode of the database.

This enumeration specifies the initialization mode of the database. The database can be created or opened.

**Enumerator**

| | |
|---|---|
| CREATE | Create a new database. |
| OPEN | Opens an existing database. |

### 4.4.3 Constructor & Destructor Documentation

#### 4.4.3.1 StorageEngine()

```
StorageEngine::StorageEngine (
            std::string directory,
            int num_bins,
            INITIALIZATION_MODE mode = INITIALIZATION_MODE::CREATE)
```

Constructs a new StorageEngine object.

**Parameters**

| | |
|---|---|
| *directory* | The directory where the key-value pairs are stored. |
| *num_bins* | The number of bins in the hash map. |
| *mode* | The initialization mode of the database. |

### 4.4.4 Member Function Documentation

#### 4.4.4.1 erase()

```
bool StorageEngine::erase (
            std::string key)
```

Erases a key-value pair from the key-value store.

**Parameters**

| | |
|---|---|
| *key* | The key of the key-value pair to erase. |

**Returns**

True if the key-value pair was erased, false otherwise.

#### 4.4.4.2 get()

```
std::optional< std::string > StorageEngine::get (
            std::string key)
```

Retrieves the value of a key from the key-value store.

**Parameters**

| | |
|---|---|
| *key* | The key to retrieve the value for. |

**Returns**

The value of the key, if it exists.

#### 4.4.4.3 insert()

```
void StorageEngine::insert (
            std::string key,
            std::string value)
```

Inserts a key-value pair into the key-value store.

**Parameters**

| | |
|---|---|
| *key* | The key of the key-value pair. |
| *value* | The value of the key-value pair. |

The documentation for this class was generated from the following file:

- include/storage_engine.h

# Chapter 5

# File Documentation

## 5.1 asio.hpp

```
//
// asio.hpp
// ~~~~~~~~
//
// Copyright (c) 2003-2024 Christopher M. Kohlhoff (chris at kohlhoff dot com)
//
// Distributed under the Boost Software License, Version 1.0. (See accompanying
// file LICENSE_1_0.txt or copy at http://www.boost.org/LICENSE_1_0.txt)
//

#ifndef ASIO_HPP
#define ASIO_HPP

#if defined(_MSC_VER) && (_MSC_VER >= 1200)
# pragma once
#endif // defined(_MSC_VER) && (_MSC_VER >= 1200)

#include "asio/any_completion_executor.hpp"
#include "asio/any_completion_handler.hpp"
#include "asio/any_io_executor.hpp"
#include "asio/append.hpp"
#include "asio/as_tuple.hpp"
#include "asio/associated_allocator.hpp"
#include "asio/associated_cancellation_slot.hpp"
#include "asio/associated_executor.hpp"
#include "asio/associated_immediate_executor.hpp"
#include "asio/associator.hpp"
#include "asio/async_result.hpp"
#include "asio/awaitable.hpp"
#include "asio/basic_datagram_socket.hpp"
#include "asio/basic_deadline_timer.hpp"
#include "asio/basic_file.hpp"
#include "asio/basic_io_object.hpp"
#include "asio/basic_random_access_file.hpp"
#include "asio/basic_raw_socket.hpp"
#include "asio/basic_readable_pipe.hpp"
#include "asio/basic_seq_packet_socket.hpp"
#include "asio/basic_serial_port.hpp"
#include "asio/basic_signal_set.hpp"
#include "asio/basic_socket.hpp"
#include "asio/basic_socket_acceptor.hpp"
#include "asio/basic_socket_iostream.hpp"
#include "asio/basic_socket_streambuf.hpp"
#include "asio/basic_stream_file.hpp"
#include "asio/basic_stream_socket.hpp"
#include "asio/basic_streambuf.hpp"
#include "asio/basic_waitable_timer.hpp"
#include "asio/basic_writable_pipe.hpp"
#include "asio/bind_allocator.hpp"
#include "asio/bind_cancellation_slot.hpp"
#include "asio/bind_executor.hpp"
#include "asio/bind_immediate_executor.hpp"
#include "asio/buffer.hpp"
#include "asio/buffer_registration.hpp"
#include "asio/buffered_read_stream_fwd.hpp"
#include "asio/buffered_read_stream.hpp"
#include "asio/buffered_stream_fwd.hpp"
#include "asio/buffered_stream.hpp"
```

```
#include "asio/buffered_write_stream_fwd.hpp"
#include "asio/buffered_write_stream.hpp"
#include "asio/buffers_iterator.hpp"
#include "asio/cancellation_signal.hpp"
#include "asio/cancellation_state.hpp"
#include "asio/cancellation_type.hpp"
#include "asio/co_spawn.hpp"
#include "asio/completion_condition.hpp"
#include "asio/compose.hpp"
#include "asio/connect.hpp"
#include "asio/connect_pipe.hpp"
#include "asio/consign.hpp"
#include "asio/coroutine.hpp"
#include "asio/deadline_timer.hpp"
#include "asio/defer.hpp"
#include "asio/deferred.hpp"
#include "asio/detached.hpp"
#include "asio/dispatch.hpp"
#include "asio/error.hpp"
#include "asio/error_code.hpp"
#include "asio/execution.hpp"
#include "asio/execution/allocator.hpp"
#include "asio/execution/any_executor.hpp"
#include "asio/execution/blocking.hpp"
#include "asio/execution/blocking_adaptation.hpp"
#include "asio/execution/context.hpp"
#include "asio/execution/context_as.hpp"
#include "asio/execution/executor.hpp"
#include "asio/execution/invocable_archetype.hpp"
#include "asio/execution/mapping.hpp"
#include "asio/execution/occupancy.hpp"
#include "asio/execution/outstanding_work.hpp"
#include "asio/execution/prefer_only.hpp"
#include "asio/execution/relationship.hpp"
#include "asio/executor.hpp"
#include "asio/executor_work_guard.hpp"
#include "asio/file_base.hpp"
#include "asio/generic/basic_endpoint.hpp"
#include "asio/generic/datagram_protocol.hpp"
#include "asio/generic/raw_protocol.hpp"
#include "asio/generic/seq_packet_protocol.hpp"
#include "asio/generic/stream_protocol.hpp"
#include "asio/handler_continuation_hook.hpp"
#include "asio/high_resolution_timer.hpp"
#include "asio/io_context.hpp"
#include "asio/io_context_strand.hpp"
#include "asio/io_service.hpp"
#include "asio/io_service_strand.hpp"
#include "asio/ip/address.hpp"
#include "asio/ip/address_v4.hpp"
#include "asio/ip/address_v4_iterator.hpp"
#include "asio/ip/address_v4_range.hpp"
#include "asio/ip/address_v6.hpp"
#include "asio/ip/address_v6_iterator.hpp"
#include "asio/ip/address_v6_range.hpp"
#include "asio/ip/network_v4.hpp"
#include "asio/ip/network_v6.hpp"
#include "asio/ip/bad_address_cast.hpp"
#include "asio/ip/basic_endpoint.hpp"
#include "asio/ip/basic_resolver.hpp"
#include "asio/ip/basic_resolver_entry.hpp"
#include "asio/ip/basic_resolver_iterator.hpp"
#include "asio/ip/basic_resolver_query.hpp"
#include "asio/ip/host_name.hpp"
#include "asio/ip/icmp.hpp"
#include "asio/ip/multicast.hpp"
#include "asio/ip/resolver_base.hpp"
#include "asio/ip/resolver_query_base.hpp"
#include "asio/ip/tcp.hpp"
#include "asio/ip/udp.hpp"
#include "asio/ip/unicast.hpp"
#include "asio/ip/v6_only.hpp"
#include "asio/is_applicable_property.hpp"
#include "asio/is_contiguous_iterator.hpp"
#include "asio/is_executor.hpp"
#include "asio/is_read_buffered.hpp"
#include "asio/is_write_buffered.hpp"
#include "asio/local/basic_endpoint.hpp"
#include "asio/local/connect_pair.hpp"
#include "asio/local/datagram_protocol.hpp"
#include "asio/local/seq_packet_protocol.hpp"
#include "asio/local/stream_protocol.hpp"
#include "asio/multiple_exceptions.hpp"
#include "asio/packaged_task.hpp"
#include "asio/placeholders.hpp"
#include "asio/posix/basic_descriptor.hpp"
#include "asio/posix/basic_stream_descriptor.hpp"
```

```
#include "asio/posix/descriptor.hpp"
#include "asio/posix/descriptor_base.hpp"
#include "asio/posix/stream_descriptor.hpp"
#include "asio/post.hpp"
#include "asio/prefer.hpp"
#include "asio/prepend.hpp"
#include "asio/query.hpp"
#include "asio/random_access_file.hpp"
#include "asio/read.hpp"
#include "asio/read_at.hpp"
#include "asio/read_until.hpp"
#include "asio/readable_pipe.hpp"
#include "asio/recycling_allocator.hpp"
#include "asio/redirect_error.hpp"
#include "asio/registered_buffer.hpp"
#include "asio/require.hpp"
#include "asio/require_concept.hpp"
#include "asio/serial_port.hpp"
#include "asio/serial_port_base.hpp"
#include "asio/signal_set.hpp"
#include "asio/signal_set_base.hpp"
#include "asio/socket_base.hpp"
#include "asio/static_thread_pool.hpp"
#include "asio/steady_timer.hpp"
#include "asio/strand.hpp"
#include "asio/stream_file.hpp"
#include "asio/streambuf.hpp"
#include "asio/system_context.hpp"
#include "asio/system_error.hpp"
#include "asio/system_executor.hpp"
#include "asio/system_timer.hpp"
#include "asio/this_coro.hpp"
#include "asio/thread.hpp"
#include "asio/thread_pool.hpp"
#include "asio/time_traits.hpp"
#include "asio/use_awaitable.hpp"
#include "asio/use_future.hpp"
#include "asio/uses_executor.hpp"
#include "asio/version.hpp"
#include "asio/wait_traits.hpp"
#include "asio/windows/basic_object_handle.hpp"
#include "asio/windows/basic_overlapped_handle.hpp"
#include "asio/windows/basic_random_access_handle.hpp"
#include "asio/windows/basic_stream_handle.hpp"
#include "asio/windows/object_handle.hpp"
#include "asio/windows/overlapped_handle.hpp"
#include "asio/windows/overlapped_ptr.hpp"
#include "asio/windows/random_access_handle.hpp"
#include "asio/windows/stream_handle.hpp"
#include "asio/writable_pipe.hpp"
#include "asio/write.hpp"
#include "asio/write_at.hpp"

#endif // ASIO_HPP
```

## 5.2 storage_engine.h

```
#pragma once
#include <string>
#include <memory>
#include <optional>

class PersistentHashMap;

class StorageEngine {
public:
    enum class INITIALIZATION_MODE {
        CREATE = 0,

        OPEN = 1
    };

    StorageEngine(std::string directory, int num_bins, INITIALIZATION_MODE mode =
INITIALIZATION_MODE::CREATE);

    ~StorageEngine();

    void insert(std::string key, std::string value);

    bool erase(std::string key);

    std::optional<std::string> get(std::string key);

private:
    std::unique_ptr<PersistentHashMap> pImpl;
};
```

# Index